High-Level Synthesis from C

- can automate C to HW synthesis using
  man techniques
- one technique is to use templates

* Assignment Statement

  \[
  \text{target} = \text{expression};
  \]

* Conditional

  ```
  if (cond) {
      \// if statements
  } else {
      \// else statements
  }
  ```
* Conditional (Continued)

```cpp
if (cond1) {
    // if statements
    else if (cond2) {
        // else if statements
    } else {
        // else statements
    }
}
```

* Loops

```cpp
while (cond) {
    // statements
}
```

```cpp
for (init; cond; incr) {
    // loop statements
}
```
Example:

Create HW for sum of absolute differences used within MPEG encoding.

```c
int SAP (unsigned char A[256], unsigned char B[256])
{
    int sum;
    int i;
    sum = 0;
    i = 0;
    while (i < 256) {
        sum += abs(A[i] - B[i]);
        i++;
    }
    return sum;
}
```

Assignment rule: Assigning a value to a variable will require one
5th transition (clock cycle) before new value can be read.
Step 1: Convert function to HLSM
- Ignore communication and memory accesses for now

Step 2: Can optimize loop to single state

- Must update condition as old value of `v` is read
Skip 3: convert array access to memory accesses

- Assume memory with two read ports and single cycle access

```c
sum = 0
i = 0
while (i < n) {
    mem_a_addr = A[i]
    mem_a_rd = 1
    mem_b_addr = B[i]
    mem_b_rd = 1
    Addr++
    B_addr++
    sum += abs(mem_a_data - mem_b_data)
    i++
}

return sum
```
- Assume memory with one read with single cycle access

\begin{algorithm}
\State \textbf{Step 1:}
\begin{enumerate}
  \item \textbf{Option 1:} Write result to memory
  \item \textbf{Option 2:} Processor reads sum from memory mapped register
\end{enumerate}
\end{algorithm}
Semantics

Assume memory requires two cycles per access (one port)

Sum := 0
T := 0
mem_addr := Addr
mem_wr := 1
Addr++

tmp := mem_data
mem_addr := Addr
mem_wr := 1
Addr++

if \( i \leq 255 \) then
Sum := Sum + abs(tmp - mem_data)
T := T + 1
else
mem_addr := Addr
mem_wr := 1
Addr++
mem_data := Sum
end